FreeLing: From a multilingual open-source analyzer suite to an EBMT platform

David Farwell and Lluís Padró
Dept. Llenguatges i Sistemes Informàtics
TALP Research Centre
Universitat Politècnica de Catalunya
Barcelona, Spain
{farwell,padro}@lsi.upc.edu

Abstract

FreeLing is an open-source library providing a wide range of language analysis utilities for several different languages. It is intended to provide NLP application developers with any text processing and language annotation tools they may need in order to simplify their development task.

Moreover, FreeLing is customizable and extensible. Developers can use the default linguistic resources (dictionaries, lexicons, grammars, etc.), or extend them, adapt to particular domains, or even develop new resources for specific languages.

Being open-source has enabled FreeLing to grow far beyond its original capabilities, especially with regard to linguistic data: contributions from its community of users, for instance, include morphological dictionaries and PoS tagger training data for Galician, Italian, Portuguese, Asturian, and Welsh.

In this paper we present the basic architecture and the main services in FreeLing, and we outline how developers might use it to build competitive NLP systems and indicate how it might be extended to support the development of Example-Based Machine Translation systems.

1 Introduction

Basic language processing such as tokenizing, morphological analysis, lemmatizing, PoS tagging, word sense disambiguation, dependency parsing, etc., is essential for most natural language processing applications such as Machine Translation, Text Summarization, Dialogue processing, and so on.

This dependence turns language analysis technologies into very valuable resources for any NLP research or development task, and the lack of availability of state-of-the-art systems constitutes a severe bottleneck to greater progress in the area, whether for research or development.

In additional, a large part of the effort required to develop NLP systems is devoted to the adaptation of existing software resources to the platform, programming language, format or API of the final application.

Thus, the objective of the FreeLing environment is to provide an environment in which basic NLP tools and resources are generally available and may be used without restrictions, so as to enhance the rapid development of advanced, more portable, NLP technologies.

We suggest that this same approach, if not FreeLing itself, can serve as a basis for developing an environment for the efficient implementation of advanced, state-of-the-art EBMT systems. Taking FreeLing as a point of departure, the focus of such an effort would be on providing greater support for the development of transfer and generation modules, for corpus alignment and mark up, and for equivalency discovery (that is to say, terminology and example induction). We assume that much of this software is already freely available from existing open-source research and development platforms and would mainly require interchange formats and possibly "rationalization," that is to say, the division of complex, interactive software components into simpler more focused utilities. By integrating such resources into an existing NLP applications development environment such as FreeLing, research and development of EBMT systems would be greatly enhanced.

2 FreeLing

In the remainder of this paper we describe version 2.1 of the FreeLing suite of NLP tools and resources. FreeLing was first released in February 2004 providing morphological analysis and PoS tagging for Catalan, Spanish, and English. From then on, the package has been improved and enlarged to cover more languages (Italian, Galician, Welsh, Portuguese and Asturian) and to offer more services: Named entity recognition and classification, chunking, dependency parsing, WordNet-based semantic annotation, UKB word sense disambiguation, and coreference resolution.

FreeLing is not conceived as an end-user oriented tool, but as library on top of which powerful NLP applications can be developed.

A remarkable feature of FreeLing is that it is distributed under a free-software GPL license, thus enabling any developer to adapt the package to his needs in order to get the most suitable behaviour for the application being developed.

Also, the library is designed to separate algorithms from linguistic data to the largest possible extent, making it possible to port most of the functionalities to new languages by simply supplying linguistic data files.

3 FreeLing 2.1 Features

Version 2.1 of the suite provides the following features:

- Tokenization.
- Sentence splitting.
- Morphological analysis, with advanced suffix and prefix handling (diminutive, appreciative, clitic pronouns, etc.).
- Date/time expression recognition.
- Currency expression recognition.
- Numerical expression recognition (numbers, quantities, percentages, ratios, etc.).
- Physical magnitude expression recognition: Speed (e.g. 120 Km/h), length (e.g. 23 cm.), pressure (e.g. 12.3 in/ft2), frequency, density, power, etc.
- Part-of-Speech tagging. Two algorithms are provided: an HMM trigram model following [4], and a relaxation labelling model based on [6] which enables the use of handwritten rules together with the statistical models.
- Named Entity detection and classification. The classification module is based on Machine Learning techniques, namely, the AdaBoost-based system winner of CoNLL'02 [5].
- Re-tokenization after PoS tagging. Some words in the Romance languages can be split once their Part-of-speech is known. For instance, the word *vela* in Spanish may be a noun (candle) but it also may mean "see her" if it is interpreted as an imperative form of the verb *ver* (to see) plus the enclitic pronoun *la* (her). The suffix handler is able to detect such cases and enrich the analysis with the relevant information. After tagging, when the category is known, the word may be split to ease syntactic analysis, or simply left whole with the information explicitly indicated.
- Chart Parser, a reimplementation of [3].
- Dependency parser, as described in [2].
- Sense annotator for English, Spanish, and Catalan based on WordNet1.6, along with most-frequent-sense word sense disambiguation.
- Seamless integration of the UKB word sense disambiguation [1] for any language having a freely available WordNet.
- Coreference resolution module based on [7].

Given the community-generated nature of the software, not all services are available for all languages. Coreference resolution and Named Entity classification are available only for Spanish. Shallow and dependency parsing is not available for Welsh, Portuguese, and Italian, and only morphological analysis is available for Asturian.

4 FreeLing Architecture

FreeLing is based on a client-server architecture, intended to support NLP developers and aiming to ease the integration of language analysis technologies into the development of higher level applications.

This architecture consists of a simple two-layer, client-server approach: A basic linguistic service layer which provides core analysis procedures (morphological analysis, tagging, parsing, etc.), and an application layer which, acting as a client, which requests the desired services from the basic service layer.

In this scenario, integrating the basic analyzers in a new NLP application is reduced to three simple steps:

- Convert the data from application internal representation to the service API data structures.
- Call the service and obtain the results.
- Convert the results to the application internal representation.

Obviously, if the application is developed natively over FreeLing, it can use its data structures in a straightforward manner, reducing the interaction to a simple function call.

The advantages of this architecture are:

- It enables to use the analysis routines as a function call from any NLP application, not as a separate software package. This is a crucial issue for modern NLP, especially for high level application development.
- The clients requesting analysis services may be not only NLP applications, but also other service providing modules (e.g. a parsing module might request a PoS tagging service). This enables the construction of increasingly more complex language analysis servers.
- Conversions are performed between client application data structures and server library data structures, being unnecessary to define data interchange formats between analyzers, and dramatically reducing the overhead caused by the reading, writing, parsing, and transmitting of text-based representations such as XML, SGML. Note that this does not mean that the client application has to adapt its input/output formats or internal representations. So long as the library is accessed via its API, the client application may handle the data at will.
- The linguistic processors do not need to be initialized for each piece of text to be analyzed.
- The application may decide how and when to invoke each analyzer, and on which text segment (i.e. there is no need for whole-text pipelined processing).
- The client-server approach enables the interaction between objects via some standard distributed object middleware (e.g. setting them as web services) which makes it possible to distribute applications over a network, activate several instances of the same service, if necessary, as well as executing on any platform client applications written in any programming language.

5 Some Internal Details

The internal architecture of FreeLing is based on two kinds of objects: linguistic data objects and processing objects.

5.1 Linguistic Data Classes

The basic classes in the library are used to contain linguistic data (such as a word, a PoS tag, a sentence, a document, etc.). Any client application must be aware of these classes in order to be able to provide to each processing module the right data, and to correctly interpret the module results.

The linguistic classes supported by the current version are:

- analyses: A tuple <lemma, PoS tag, probability, senses>.
- words: A word form with a list of possible analysis.
- sentences: A list of words known to be a complete sentence, it may include also a parse tree and/or a dependency tree.
- paragraphs: A list of sentences known to be an independent paragraph.
- documents: A list of paragraphs that form a complete document. It may also contain coreference information about the entity mentions in the document.

Figure 1 presents a UML diagram with the linguistic data classes.

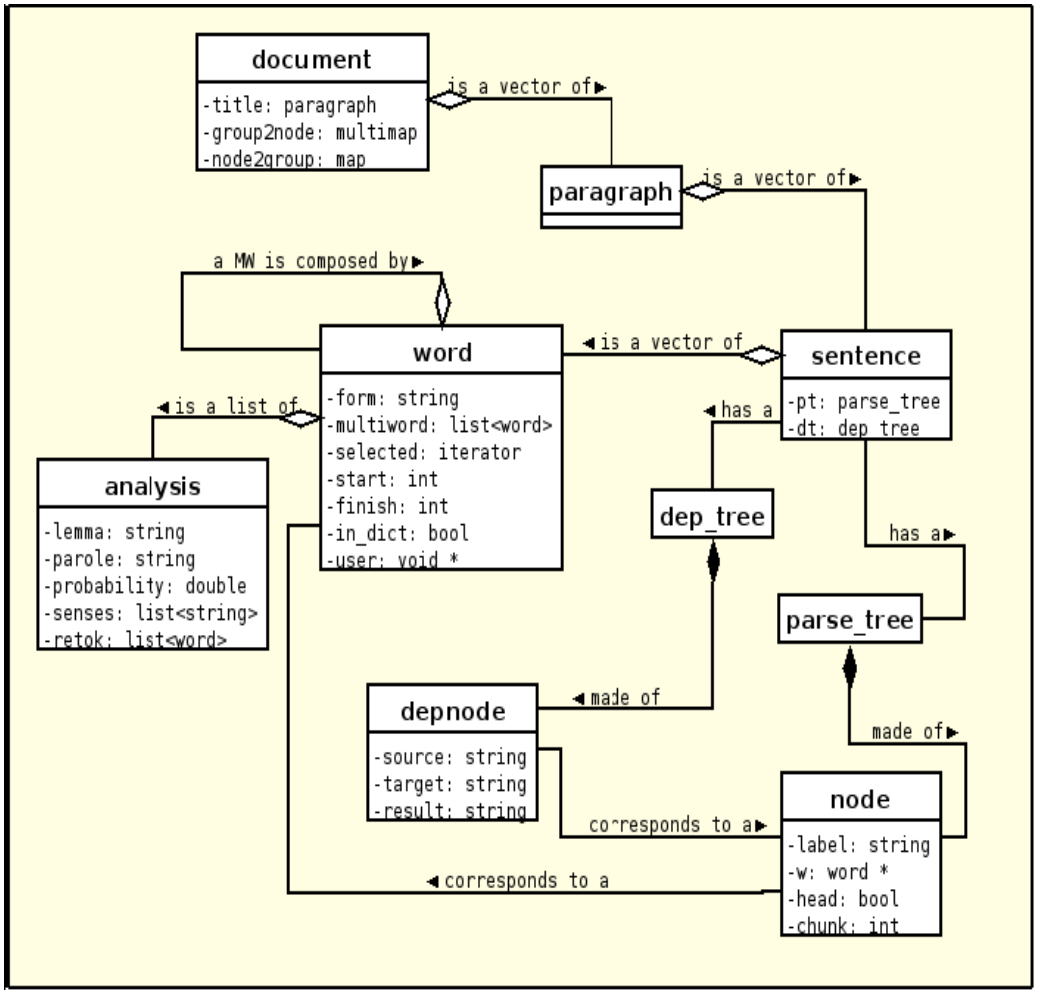


Figure 1: FreeLing-2.1 Linguistic Data Classes.

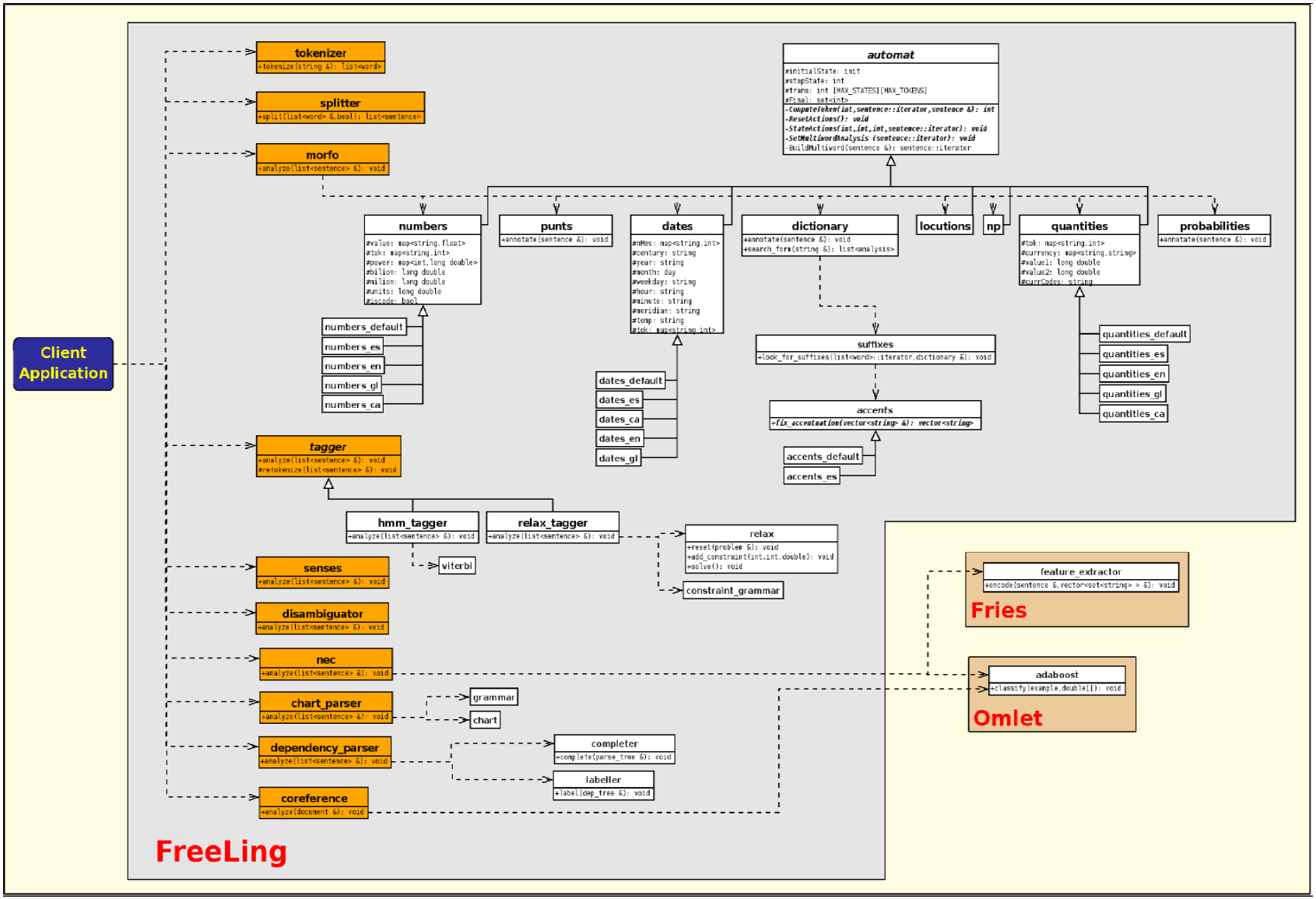
5.2 Processing Classes

Apart from classes containing the linguistic data, the library provides classes able to transform this data:

- tokenizer: Receives plain text and returns a list of word objects.
- splitter: Receives a list of word objects and returns a list of sentence objects.
- morfo: Receives a list of sentences and morphologically annotates each word object in each sentence. In fact, this class applies a cascade of specialized processors (number detection, date/time detection, multiword detection, dictionary search, etc.) each of which is in turn a processing class:
  - locutions: Multiword recognizer.
  - dictionary: Dictionary lookup and suffix handling.
  - numbers: Numerical expressions recognizer.
  - dates: Date/time expressions recognizer.
  - quantities: Ratio and percentage expressions and monetary amount recognizer.
  - punts: Punctuation symbol annotator.
  - probabilities: Lexical probabilities annotator and unknown words handler.
  - np: Proper noun recognizer.
- tagger: Receives a list of sentence objects and disambiguates the PoS of each word object in each sentence. If the selected analysis carries re-tokenization information, the word may be split in two or more new words.
- NE classifier: Receives a list of sentence objects and classifies all word objects tagged as proper nouns in each sentence.
- Sense annotator: Receives a list of sentence objects and enriches with synset information the analysis chosen by the tagger for each word object.
- Word sense disambiguator: Receives a list of sentence objects and ranks the possible senses for the analysis chosen by the tagger for each word object.
- chunk parser: Receives a list of sentence objects and enriches each of them with a parsed tree object.
- dependency parser: Receives a list of parsed sentence objects and enriches each of them with a dependency tree object.
- coreference solver: Receives a document formed by parsed sentence objects and enriches the document with coreference information.

Figure 2 below presents a UML diagram with the processing classes.

The client application is free to decide in which format to input, output or store its linguistic data, and only needs to translate it into the classes described above when interacting with the library. Also, the client application is free to decide which processing steps the library is going to be used for –e.g. the application may require a tagger for Spanish but not for Catalan, or it may want to call the morphological analyzer directly skipping tokenization and sentence splitting, or it may only want to apply a date/time expressions recognizer, without using any other functionality, and so on.


Client Application
tokenizer
+tokenize(string &): list<word>
splitter
+split(list<word> &,bool): list<sentence>
morfo
+analyze(list<sentence> &): void
automat
#initialState: int
#stopState: int
#trans: int [MAX_STATES][MAX_TOKENS]
#final: set<int>
-ComputeToken(int,sentence::iterator,sentence &): int
-ResetActions(): void
-StateActions(int,int,int,sentence::iterator): void
-SetMultiwordAnalysis (sentence::iterator): void
-BuildMultiword(sentence &): sentence::iterator
numbers
#value: map<string,float>
#tok: map<string,int>
#power: map<int,long double>
#billon: long double
#million: long double
#units: long double
#iscode: bool
punts
+annotate(sentence &): void
dates
#mMes: map<string,int>
#century: string
#year: string
#month: day
#weekday: string
#hour: string
#minute: string
#meridian: string
#temp: string
#tok: map<string,int>
dictionary
+annotate(sentence &): void
+search_form(string &): list<analysis>
locutions
np
quantities
#tok: map<string,int>
#currency: map<string,string>
#value1: long double
#value2: long double
#currCodes: string
probabilities
+annotate(sentence &): void
numbers_default
numbers_es
numbers_en
numbers_gl
numbers_ca
dates_default
dates_es
dates_ca
dates_en
dates_gl
suffixes
+look_for_suffixes(list<word>::iterator,dictionary &): void
accents
+fix_accentuation(vector<string> &): vector<string>
accents_default
accents_es
quantities_default
quantities_es
quantities_en
quantities_gl
quantities_ca
tagger
+analyze(list<sentence> &): void
#retokenize(list<sentence> &): void
hmm_tagger
+analyze(list<sentence> &): void
viterbi
relax_tagger
+analyze(list<sentence> &): void
relax
+reset(problem &): void
+add_constraint(int,int,double): void
+solve(): void
constraint_grammar
senses
+analyze(list<sentence> &): void
disambiguator
+analyze(list<sentence> &): void
nec
+analyze(list<sentence> &): void
chart_parser
+analyze(list<sentence> &): void
grammar
chart
dependency_parser
+analyze(list<sentence> &): void
completer
+complete(parse_tree &): void
labeller
+label(dep_tree &): void
coreference
+analyze(document &): void
FreeLing
feature_extractor
+encode(sentence &,vector<set<string> > &): void
Fries
adaboost
+classify(example,double[]): void
Omlet

6 Extending FreeLing to support EBMT

While what counts as EBMT is not immediately obvious, a classic model, which grew out of the need to improve the non-fluent target language output of rule-based Japanese-English translation systems, is based on a standard transfer approach between syntactic dependency representations for source and target languages. Roughly the idea is to match the input string against the source language expressions of an example-base and, if a match is successful, the example, syntactically annotated, is used to identify the target language equivalent, thereby circumventing the actual application of the rule base. If no match is found, then the input is analysed using the rule based and the results of the two procedures are merged. In general, the more the translation process is able to take advantage of the example base, the more accurate and fluent the translation.

Given this classic model, then in addition to the basic utilities for language analysis described above, additional utilities to support the development of transfer and generation procedures will need to be integrated into the FreeLing environment. Many of these utilities may already be publically available (e.g., from Apertium, the NLG group at the Open University, etc.), particularly for the relatively shallow representations typical of MT. They need only be adapted to the FreeLing environment. Given the general trend today toward the development of open-source software, other utilities no doubt may be added as they become available. Still others might be taken from existing open-source transfer and generation components or systems by decomposing them into their component procedures.

Just as importantly, the FreeLing environment must be extended to support the development of bilingual and multilingual corpus annotation procedures as well as the development of discovery procedures for finding translation equivalents, that is to say, bilingual or multilingual term banks and example bases. Indeed, under one interpretation of EBMT, the source language expression is segmented entirely on the basis of the source language expressions in the example base without any recourse to syntactic representation. In such cases, example expressions often consist of single word forms and so the basic approach is essentially one of direct substitution. If the techniques for constructing the example base and for matching the input against the example base are statistical, there is little to distinguish EBMT from a standard phrase-base statistical MT. Thus, an important extension to the existing FreeLing environment will include corpus alignment and translation model induction procedures. Again, much of this software is already freely available from existing open-source research platforms (e.g., Giza, Moses, openTMS, etc.) and will mainly require adaptation to the FreeLing environment.

Finally, access to publically available bilingual and multilingual resources such as dictionaries, term banks, gazetteers, language memories and at least some core parallel bilingual and multilingual corpora must be made accessible as well. Of particular interest here are the Europarl, the Canadian Hansard, and Hong Kong Hansard corpora although additional corpora may be included from the outset as well. Beyond these there is a wealth of publically available parallel corpora which could be integrated as needed as well into the FreeLing environment in a relatively straightforward manner. In time, depending on the contributions of the user community, annotated corpora should also be made available.

7 Conclusion

With extensions such those sketched above, FreeLing, or some similar NLP development platform, could function as a productive open-source environment for the development of state-of-the-art EBMT systems, providing the basic utilities for the implementation of a wide range of EBMT software and resources. As such, it would significantly reduce the software bottleneck faced by EBMT developers today and make an important contribution to the long term success of example-based machine translation.

References

[1] Eneko Agirre and Aitor Soroa. Personalizing pagerank for word sense disambiguation. In Proceedings of the 12th conference of the European chapter of the Association for Computational Linguistics (EACL-2009), Athens, Greece, 2009.
[2] Jordi Atserias, Elisabet Comelles, and Aingeru Mayor. Txala un analizador libre de dependencias para el castellano. Procesamiento del Lenguaje Natural, (35):455–456, September 2005.
[3] Jordi Atserias and Horacio Rodríguez. Tacat: Tagged corpus analyzer tool. Technical report LSI-98-2-t, Departament de LSI. Universitat Politècnica de Catalunya, 1998.
[4] Thorsten Brants. Tnt - a statistical part-of-speech tagger. In Proceedings of the 6th Conference on Applied Natural Language Processing, ANLP. ACL, 2000.
[5] Xavier Carreras, Lluís Màrquez, and Lluís Padró. Named entity extraction using adaboost. In Proceedings of CoNLL Shared Task, pages 167–170, Taipei, Taiwan, 2002.
[6] Lluís Padró. A Hybrid Environment for Syntax–Semantic Tagging. PhD thesis, Dept. Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya, February 1998. http://www.lsi.upc.es/˜padro.
[7] W.M. Soon, H. T. Ng, and D.C.Y. Lim. A machine learning approach to coreference resolution of noun phrases. Computational Linguistics, 27(4):521–544, 2001.